(2)

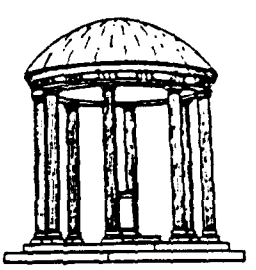
Incremental Volume Rendering Algorithm for Interactive 3D Ultrasound Imaging

TR91-003
February, 1991

Ryutarou Ohbuchi
Henry Fuchs

N00014-86-K-0680

Medical Image Display Group
Department of Computer Science
The University of North Carolina
Chapel Hill, NC 27599-3175



The research reported herein was carried out with the partial support of NIH grant number P01CA47982.
To appear in Proceeding Information Processing in Medical Image (IPMI XII), 1991.

UNC is an Equal Opportunity/Affirmative Action Institution.

Incremental Volume Rendering Algorithm for Interactive 3D Ultrasound Imaging

Ryutarou Ohbuchi
Henry Fuchs

Department of Computer Science,
University of North Carolina at Chapel Hill,
Chapel Hill, NC 27599,
U.S.A.

Thursday, February 7, 1991

Introduction

We have been working toward a medical 3D ultrasound scanner system that will acquire and display a 3D volume in real-time. We call this system an 'ultimate' 3D echography (3DE) system. It will acquire, display, and manipulate a 3D volume image in real-time. Such real-time-ness can be crucial for an application such as cardiac diagnosis, where single time-slice 3D image or even a dynamic display of single image acquired over many cardiac cycles by gated acquisition might miss certain kinetic features. The 'ultimate' system has two major components, the image acquisition part and the image visualization part. We at UNC-Chapel Hill have been working on the real-time 3D visualization part of the system, while Dr. Olaf von Ramm's group at Duke University has been working on the real-time 3D image acquisition part of the system.

To study various issues involved in 3D ultrasound image visualization necessary for the 'ultimate' system before the real-time 3D ultrasound image acquisition system becomes available, we have been developing an *incremental, interactive 3DE* scanner system that will acquire and display the 3D image incrementally at an interactive rate. The system will use a state-of-the-art medical real-time 2D ultrasound echography scanner as an image input, where a user guided scanhead is tracked with 3 degrees of freedom. Using the geometric information acquired for each 2D image slice, the system incrementally reconstructs the 3D array of sample points spaced regularly on the Cartesian coordinate from a stream of 2D slices located and oriented arbitrarily with 3 degrees of freedom. Reconstructed volume is then rendered, again incrementally, to produce volume rendered 2D images. To offer high degree of interactivity to the user, this visualization is done incrementally so that every new 2D slice acquired affects the final image promptly.

We present here the brief introduction to the incremental, interactive 3DE system, and the incremental volume rendering algorithm designed for high degree of interactivity. First, we will briefly describe the 'ultimate' system, then present a review of previous works on 3D ultrasound imaging. It is followed by

the sketch of the incremental acquisition and visualization system we have been working on. We will then describe the incremental volume rendering algorithm in detail. A new ray-clipping algorithm to make ray-sampling faster, called *D-buffer* algorithm is presented, which is followed by a proposed enhancement to the idea of ray-caching called *hierarchical ray-caching* for faster compositing. Then, another algorithm is proposed which combines D-buffer with the hierarchical ray-caching to provide fast integrated rendering of polygons, polyhedron defined volumes for cutaways, etc. to provide fast interaction with the volume image.

The 'Ultimate' 3D Ultrasound Scanner

Among various medical imaging modalities, ultrasound echography is the closest to achieving 3D real-time acquisition, even though other modalities such as MRI is becoming faster than before. To acquire real-time 3DE image, the 'ultimate' 3DE system will use a new scanner being developed by Dr. Olaf von Ramm's group at Duke University [Shat84]. Due to the velocity of sound limitation (about 1540m/s in water), scanning a 3D volume with reasonable resolution (128x128x128 or more) in real-time (30 3D-frames/s) requires parallel processing. The new scanner will use a single-transmit/multiple-receive scheme called *Explososcan* to increase data acquisition bandwidth. The first implementation will use a 16x16 2D-array transducer along with 16x16x64 digital delay-lines implemented in VLSI chips for 3D beam steering and focusing with 64-way multiple simultaneous reception.

Once the real-time 3DE data has been obtained, the remaining issue is the visualization of such a data. We at UNC-Chapel Hill have been working on this problem. One major effort is the display system based on the volume rendering technique for visualization that will display 3DE data in real-time. Such a display system has to cope with the challenge of very high data bandwidth. The real-time 3DE scanner above will produce on the order of 2-4 Million points per frame, or about 60-120 Million points per second. Visualizing a 3D volume data of this bandwidth in real-time requires very large computational power, on the order of Giga floating point operations per second, if straightforward algorithm is used. We are approaching this issue through the parallelism, and algorithm efficiency gained by exploiting various forms of coherences.

Effective visualization method is another major issue. It involves standard problems associated with visualization of 3D data, such as obscuration. To look at the inside of the left ventricle, images of fat tissue, part of myocardium, etc., in front must be removed to reveal the object behind. This usually requires extensive manipulation of the 3D image data. On top of those are the additional difficulties characteristic to the ultrasound data, e.g., speckle noise, gain variation, shadowing, specular artifact, etc., that can hinder the visualization. On this end of the problem, Wei-jyh Lin at UNC-Chapel Hill has been working on the algorithm to estimate surfaces in 2D echography image under the presence of speckle noise using multi-scale filtering technique. This surface estimator will be joined as a front-end to the display system described above in the future.

Previous Work in 3D Ultrasound Imaging

There have been many works that aimed at what can be termed as 3D ultrasound echography. Most of the studies known to the author have tried to reconstruct 3D data out of imaging primitive data of lesser dimension (usually 2D image), instead of more or less directly capturing 3D data. This is due to the obvious lack of 3D scanner that can acquire 3DE image in a (real-time) scan. Further, most of them used 2D echography image with uniform sampling interval in Cartesian coordinate as their primitive for image acquisition; none used 0D or 1D primitives.

The location and orientation of imaging primitives must be available to reconstruct 3DE data. Coordinate values are either explicitly tracked, as in [Brin78] [Ghos82] [Hott89] [Raic86] [McCa88] [Nikr84] [Stick84] [Mills90] using mechanical, acoustic, optical tracking mechanisms, or it is controlled implicitly at the time of acquisition [Lalo89] [Naka84] [Bill90]. One of the most interesting recent work in 3D ultrasound echography acquisition is a near-real-time, automatic 3D scanner system [Bill90]. This system is the closest yet to the real-time 3D ultrasound scanner, and is being developed at Phillips Paris Research Lab., following the earlier work [Hott89] which was a manual guided scanner with mechanical tracker as our research. This near-real-time 3D scanhead is a 'double wobbler' mechanical sector scanner, where a conventional wobbler 2D sector scanhead is rotated, or wobbled, in an additional axis by a stepping motor to provide 3D scanning. In about 3 to 5 seconds period, about 50 to 100 slices of 2D sector scan image can be acquired.

For presenting the scanned result, there are two forms; non-visual, and visual. The latter can be classified further by the rendering primitives, which is either (geometric) graphic, or image. The majority of the earlier studies [Brin78] [Ghos82] [Raic86] [Nikr84] [Stick84] had non-invasive estimation of the volume of the heart chamber as their primary objective. Thus, often the reconstruction is only *geometric*. A typical process involved a manual tracing of the pictures of 2DE images taken from the video-taped images using a digitizer. Since the visual presentation is a secondary matter, these studies use simple rendering of the geometrical reconstruction result using wire frames or a stack of contours.

More recent studies by [Naka84] [Lalo89] [McCa88] [Hott89] [Bill90] actually reconstructed 3D grey level *images*, preserving the grey scale image, which can be crucial to such purpose as a tissue characterization. [Lalo89] is a mammogram study using a special 2DE scanner which can acquire and store 45 consecutive parallel slices with 1mm interval. It is reconstructed by cubic-spline interpolation, and volume rendered. [McCa88] performed the gated acquisition of a heart's image over a cardiac cycle. They used a video-tape to store 2DE images and the volume rendering to generate images. Upon reconstruction, 'repetitive low-pass filtering' is done on the 3D volume image to suppress aliasing artifact by filling spaces between radial slices. [Bill90] uses re-slicing of the volume data by an arbitrary plane as the primary display mode for its interactive response on a current workstation (a SUN4). The system also has volume rendering as an option, in which case manual segmentation of image slices to expose volume of interest is often involved. The reconstruction algorithm is a straightforward low-pass filtered reformatting of manually selected slices.

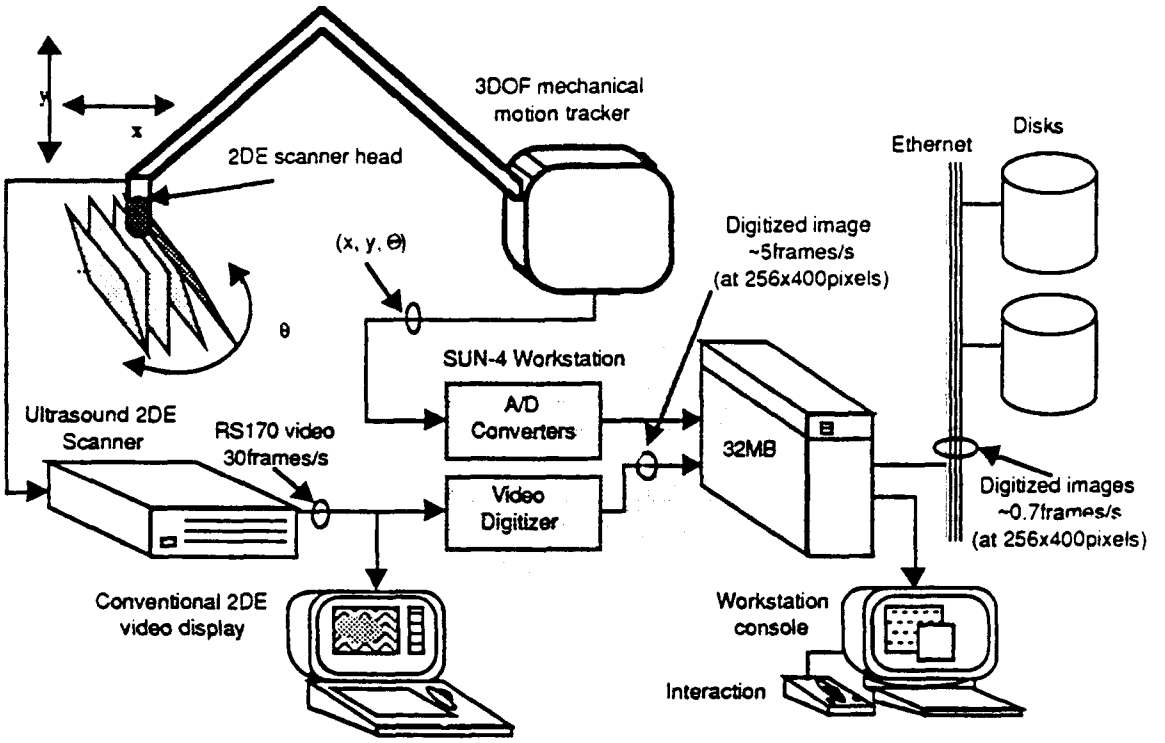


Figure 1. Incremental 3D volume data acquisition system using conventional 2D echography scanner

Incremental, Interactive 3D Acquisition and Visualization

To study the issues of real-time ultrasound 3D echography visualization before the real-time 3DE acquisition system becomes available, we have been studying an *incremental, interactive 3D echography (3DE) system*. In this system, a user-guided scanhead mounted on a 3 degree of freedom (3 DOF) mechanical tracking apparatus will acquire a series of 2D image slices as well as the corresponding goemetries, i.e., location and orientation, of each slice. Using these geometries, a regular 3D volume data where sample points are at uniform intervals in Cartesian coordinate is reconstructed from a series of 2D images with irregular geometries. This reconstruction process and the following volume rendering process take place incrementally, as each new 2D image slice arrives. Each new 2D image will affect the final rendered image promptly without waiting for the rest of the slices to arrive. (Since the scanning may continue for an indefinite period of time, possibly sweeping the same volume many times, waiting for the 'rest' loses its meaning.)

Figure 1 shows the image acquisition system for the incremental, interactive ultrasound scanner system (See Figure 2 for the picture). The 2DE scanner mounted on a mechanical tracking arm with 3 DOF acquires 2D images frames at a maximum rate of around 30 frames/sec. Each 2DE image slice from the ultrasound scanner is video-digitized in real-time by Matrox MVP-S video digitizer board, and copied into SUN-4 workstation. This copying process

is relatively slow, due to the MVP-S's frame buffer design, and runs at 5 frames/s for 256x400 pixel images.

The potentiometers in the 3D tracking arm transduce the coordinate and orientation (x, y location and angle θ) of each of the image frames through Data

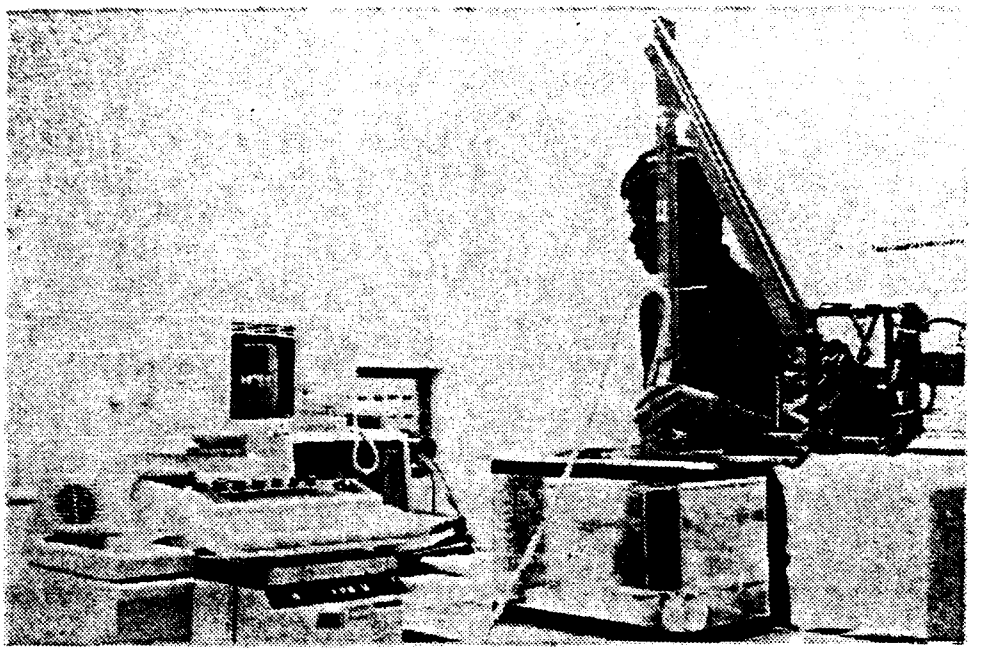


Figure 2. Scanning setup for the incremental, interactive 3D ultrasound scanner system.

Figure 3a. (Left) The doll phantom. The height is about 18 cm.

Figure 3b. (Right) A 2D echography image of the leg section of the doll phantom.

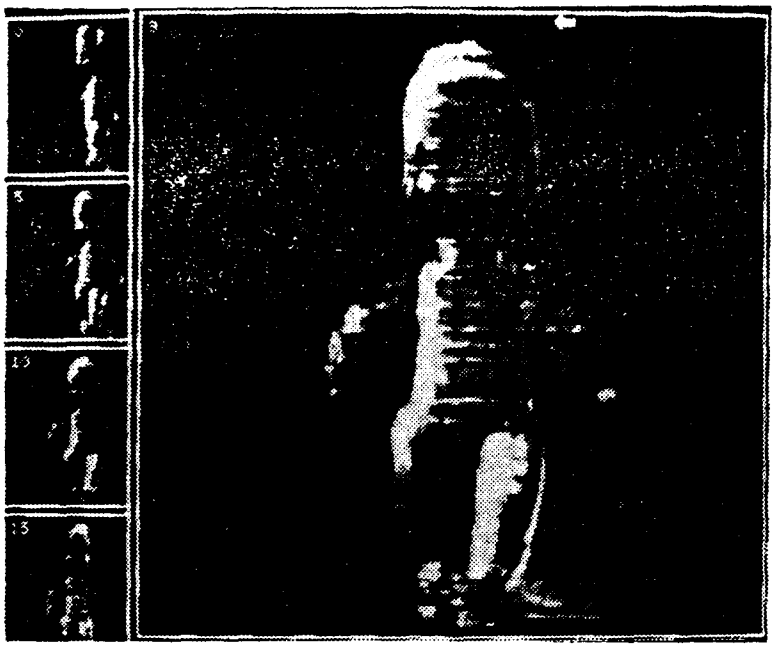


Figure 4. Reconstructed and volume rendered image of the phantom.

Translation DT-1401 A/D converter board. The tracking system has the accuracy of about 1mm in x-y position and the maximum sampling rate of around 800 sample/s. The arm came from the previous generation 2D echography scanner Rohe ROHNAR 5580. ROHNAR 5580 used the tracking to generate 2D image from a 1D scanhead. In our system, the tracking is used to generate 3D volume from a state of the art 2D scanner.

Upon acquisition, to regulate the spatial sampling rate to some degree, the acquisition of image and geometry can be made only if the change in geometry exceeds certain preset threshold. Since the scanning of the 3D volume takes some time, the object must be stable enough. Examples of the possible imaging targets are a liver or women's breast with possible immobilization. Scanning a moving target, such as a beating heart is not considered as an objective.

This acquisition setup is similar to the work at the Paris Phillips Research Lab. by François Hottier et al [Hott89], which used a similar mechanical tracking arm. In their work both scanned 2D image and the scanhead tracking result are stored in VCR and in PC's disk, respectively, for later off-line selection of the images and their matching with the geometry values. We have used the real-time video digitizer and the tracking arm to acquire both image and geometry at the same time.

Acquired 2DE image slices are incrementally reconstructed by local interpolation into regularly sampled 3D volume data by merging the new data with the existing volume data. As the slices accumulate into 3D volume, the rendering takes place, which shows the build-up of 3D volume. We will describe the details of the incremental volume rendering algorithm later in this paper. One thing of note is that the volume can be scanned repeatedly, to get better spatial sampling or to gain better acoustic window (e.g., to avoid bones). The reconstruction algorithm takes these modes of use into consideration, and provides reconstruction buffer update policy that includes 'complete replace' and 'replace by weighted average'.

Currently, images and coordinates are stored into disk of a file server connected by Ethernet for later reconstruction and rendering experiments on workstations such as DECStation 3100. In the 1991, we hope to demonstrate a system that will use a powerful graphics multicomputer Pixel-Planes 5 [Fuch89] for parallel reconstruction and rendering. In this parallel system, incremental visualization process is expected to take place at an interactive rate of more than a frame per second without ever storing the image into the disk. As mentioned, image read-out from the video digitizer is slow, and this might limit the processing speed of the entire system to 5 frames/s acquisition rate. Assuming 4mm elevation resolution of the scanner image slice, target volume of 20 cm thickness can be scanned as 100 slices of equally spaced parallel planes if sampled at Nyquist sampling rate of 2 mm. Scanning the volume with 50-100 slices will take 1.4-3 s assuming 30 2D-frames/s acquisition rate, while 10-20 s assuming 5 2D-frames/s acquisition rate.

We have conducted a preliminary data acquisition and rendering experiment, which is reported in [Ohbu90]. We used ATL Mark-4 scanner with 3.5 MHz linear scanhead as the image input, and scanned a phantom (doll of a baby, in Figure 3a) and a human forearm in a water bath. Figure 3b shows the 2D echigraphy image of the doll phantom. Figure 4 shows the reconstructed and rendered image of the doll from 90 slices of roughly parallel, roughly 2mm interval 2D image slices. It is incrementally reconstructed using forward-mapping octahedral kernel reconstruction algorithm, which will be explained later.

Incremental Volume Visualization

In this section, algorithm for the incremental, interactive visualization of 3DE images from a sequence of 2DE slices is presented. The visualization stages of the incremental, interactive 3DE system can be divided into two major stages, *reconstruction* and *rendering*. The emphasis is on the incremental volume rendering algorithm, for the reconstruction stage still requires much work.

Figure 5 shows the pipeline of the incremental volume visualization. Video digitized 2DE image slices from the scanner are incrementally *reconstructed* into a 3D scalar field sampled at uniformly spaced 3D grid points on a Cartesian coordinate system. Reconstruction is done into a *reconstruction buffer*, which resides in the world coordinate. It is then rendered using a modified front-to-back image-order volume rendering algorithm as developed by Levoy [Levo89]. Reconstructed 3D scalar field in the reconstruction buffer is *classified* (non-binary classification) and *shaded* to result 3D *shade buffer* which also sits in the world coordinate. Next steps are the ray-casting process, where sample points in the world coordinate are transformed into 3D screen coordinate and then composed into 2D screen coordinate to yield a volume rendered image.

Incremental Volume Reconstruction

Several different approaches are possible to visualize irregularly sampled volume data. Theoretically, a stream of 2D image slices with variable geometry can be rendered directly by a certain rendering algorithm. Some of the

algorithms to directly render irregular data samples have appeared recently [Garr90] [Wilh90] [Miya90] [Neem90] [Shir90].

We have chosen to reconstruct a regularly sampled 3D volume data in the *reconstruction buffer* which is then rendered using more or less conventional volume rendering algorithms that expect volume data sample at regular 3D grid points. One reason for this is the speed of rendering. Once reconstructed, the volume data sampled on regular 3D grid stored in 3D array with implicit geometry and connectivity offers advantage in accessing the data. Another reason is that we wanted to allow multiple scanning sweeps of the target volume by the scanner which is merged into single image. Achieving this with direct rendering requires storing large, unknown number of input image slices as they are, and this is not memory efficient nor computationally efficient.

In designing the visualization scheme, we assumed that the target scalar field being sampled is continuous. Thus, if the target volume is sampled with sufficient sampling rate by the 2D image slices, the resulting image of the target volume generated from these 2D slices must be continuous. For example, the resulting image should not look like a set of intersecting image planes with empty spaces in between. This means that the reconstruction requires some form of interpolated resampling. But we have to be careful about interpolation. If the sampling rate is not sufficient, there is no way of recovering the unsampled information. To not to 'lie' to the user by artificial

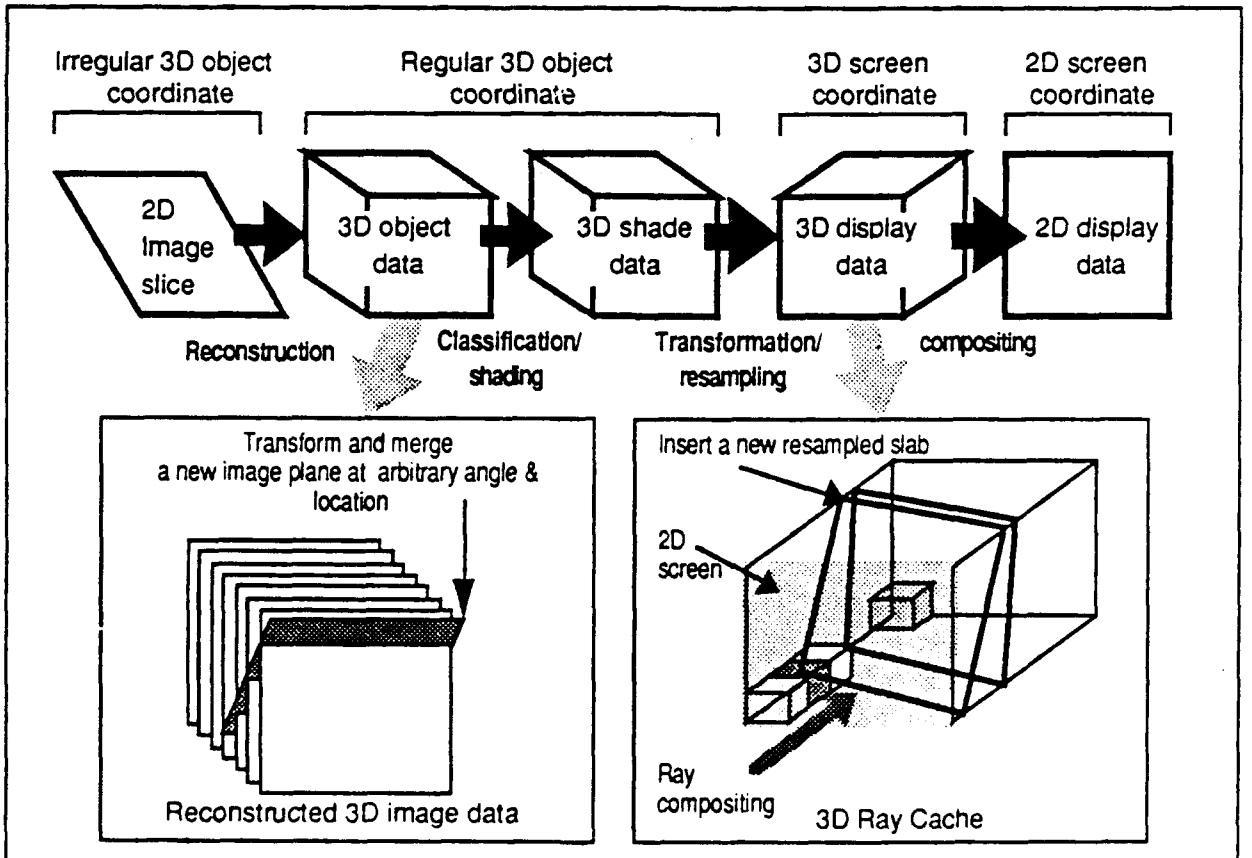


Figure 5. Incremental volume visualization processing stages. Irregular data slices are first reconstructed into a regularly sampled 3D volume data. The volume data go through standard volume rendering process, but the processing is limited to inside the slab formed by last two image slices.

data, the user should be notified by some means of the fact that it is undersampled at acquisition.

We try to reconstruct a 3D array of sample points at regular intervals on Cartesian coordinate from a series of 2D image slices with 3 DOF as the imaging primitive. We assume that there is a rectangular volume in the real-world we are imaging which will be reconstructed to the 3D sample points uniformly spaced on the 3D Cartesian coordinate. Sample points on 2D image slices that are outside of this rectangular volume will be discarded.

Next question is the interpolation techniques to use. One important characteristic of our visualization scheme is that each input image should promptly be reflected in the final image. Thus, the interpolation method used for reconstruction can not be what Franke [Fran82] call 'global' method of interpolation where the interpolant is dependent on all data points. Both of the two reconstruction methods we have implemented so far are 'local' kind.

Ideally, the interpolation function should depend on the imaging system. We have discovered, from the images obtained by scanning 3D geometric calibration phantom made of thin wires and beads, that the image slice is fairly thick. A 2mm beads is visible on the screen after 4 to 5 mm translation of the scanhead in elevation direction. This suggests much lower elevation resolution than the range resolution. Also, as expected, the sampling function in the far view is fuzzier and larger than the near view. Though this has not bee quantitative, this gave us the feel of the shape of the sampling functions to expect.

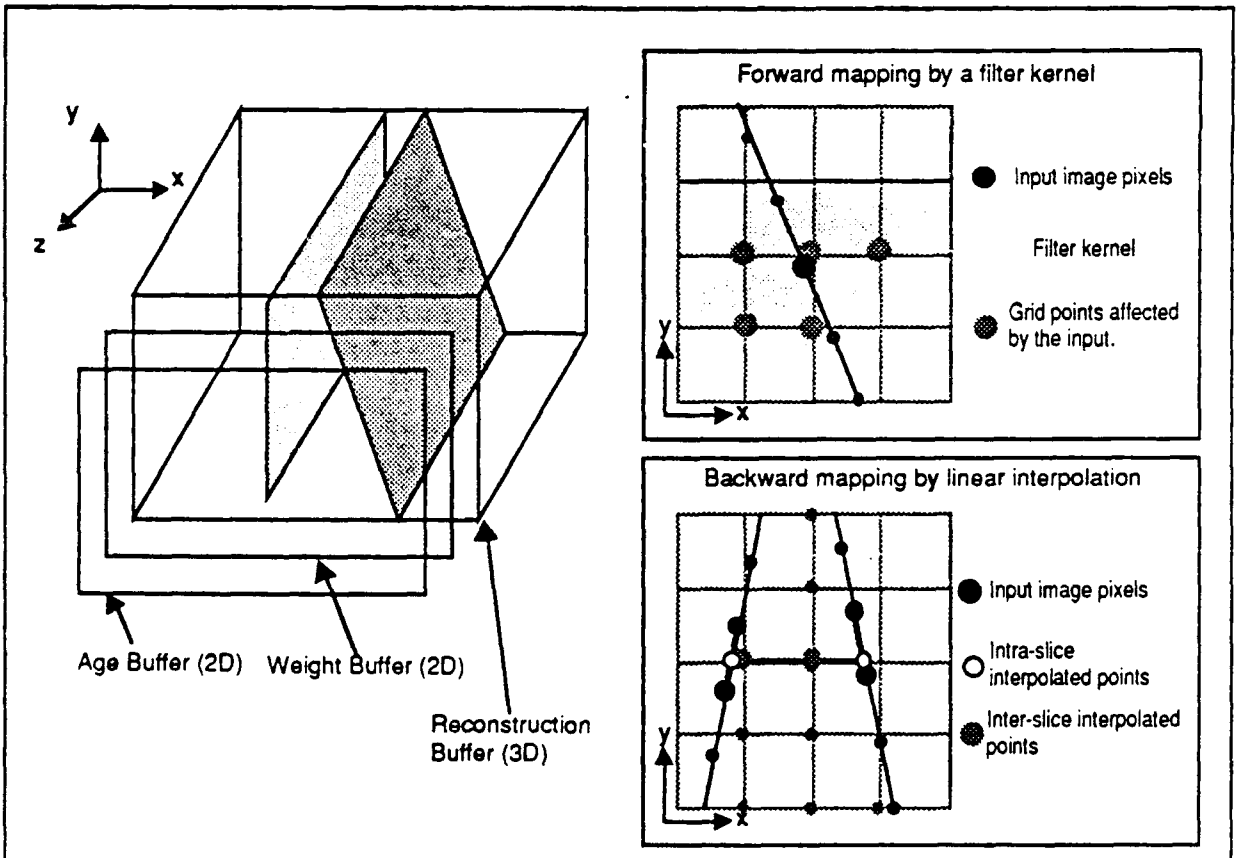


Figure 6. Reconstruction buffer and two reconstruction algorithms implemented: forward mapping using small filter kernel (above) and backward mapping algorithms.

We are still experimenting with the reconstruction algorithm. One of the method we have tried is a forward mapping algorithm, where the input sample points in the 2D image slices are distributed using a small spatial filter kernel, to the reconstruction buffer grids around the input sample points (Figure 6). The kernel is anisotropic, where its size is determined in the input parameter file to the program. Mostly, we have used a filter kernel of octahedral shape in 3D with the long-axis of around 6 mm and two short-axes of around 3 mm, which resembles the estimated shape of the ultrasound scanner system's sampling function. This corresponds to 12.5 x 6.2 x 6.2 voxels in the reconstruction buffer. Since the sampling function is attached to the image slice's coordinate, the kernel rotates along with the input slice. For each orientation of the slice, coefficients of the octahedral kernel are computed on the fly on the discretized x-y grids of the voxels. The 2D weight buffer in the diagram accumulates the weight of the kernel, for later normalization. The age buffer records the 'staleness' of the image in the reconstruction buffer, to help determine the weight to mix the value in the buffer with the new contribution.

Another reconstruction algorithm is a backward mapping kind, where the algorithm steps in the reconstruction buffer grid, while the nearby sample values from the last two input image slices are linearly interpolated and collected. This is only C^0 continuous, and its C^1 discontinuity tends to show up in the image by gradient approximation operator. Though it is fast, this algorithm does not produce as good a result as the one above.

We will continue to do more research in this area to find satisfactory reconstruction with small enough computational cost for interactive visualization.

Incremental Volume Rendering

Once the volume data with regular 3D sample points is reconstructed, it can be rendered using a standard volume rendering algorithm as described in the literatures [Upso88] [Sabe88] [Levo88]. As is known, volume rendering can be computationally expensive. The cost of image generation from reconstructed data must be made small enough to achieve the goal of interactive image generation rate on a moderate scale hardware.

The incremental volume rendering algorithm tries to reduce computation by taking advantage of three assumptions; 1) 2DE image slices are acquired incrementally, 2) shading parameters will not change for every few frames, 3) viewpoint will not change every few frames. If these conditions are met, an incremental rendering can be done. By incrementally shading and ray-sampling per reconstructed slab, computation is limited to the 'slab' formed by the last two inserted images, instead of the entire volume of the reconstruction buffer. Under the assumption of fixed viewpoint, incremental ray-sampling is realized by caching the tri-linear interpolated ray samples from the previous slabs in a 3D array in the 3D screen space called *ray cache*. This way, expensive ray-sampling is decoupled from relatively inexpensive ray-compositing, which is performed essentially for its entire ray span inside the reconstruction buffer for each frame.

Figure 5 shows the process of incremental volume rendering. After reconstruction, classification is done by table look up, and then the shading values, color and opacity, are computed. Shading takes place incrementally,

only inside the slab formed by the current and previous image slices. Currently, two shading algorithms are implemented; 1) image value shading which directly maps from input scalar value to the color value by table look up, and 2) gradient-Phong shading which performs Phong shading with diffuse and specular components, where surface normal is approximated by finite difference. Shading method 2) can emulate 1) by setting the look-up table appropriately, but it is included for its efficiency. Simple but fast X-ray like projection which result from this, if combined with additive compositing, can be quite useful in rotating the object to find best viewpoint. The shading result is stored in the 3D shade buffer that resides in the world coordinate.

The ray-sampling stage performs actual ray-casting, where a ray is cast from each pixel into 3D shade buffer, and the shading values are sampled at uniform interval along the ray. We used the perspective projection in the hope of giving better 3D perception. Perspective projection can have over and/or under sampling problems in ray-casting, but no measure such as the one found in [Novi90] is included in the current implementation. A ray sample is the result of tri-linear interpolation from eight points surrounding the sample point to minimize aliasing in the resulting images. This stage also works incrementally, and only the same sub-volume processed in the shading stage is sampled. To sample only inside the slab, rays are clipped to the slab by an algorithm that clips a line to a convex polyhedron (Figure 7a). The slab, if it happens to be concave (Figure 7b) is first decomposed into two convex polyhedrons. To clip rays to the polyhedron(s), we developed an algorithm which turned out to be essentially the same as the Cyrus-Beck clipping algorithm [Cyru78].

Ray sample values are saved in a 3D array called *ray-cache* in 3D screen space. This is a classic space-time trade off. Each pixel in the frame buffer is associated with a linear array of ray samples along the ray. As the new slab is shaded and sampled, those samples are inserted to the appropriate locations (depth) in the ray-cache, replacing the old value. All the other locations are not sampled and their values stay the same. Current implementation has multiplicative compositing as in [Levo88] as well as additive compositing for X-ray like image, and Maximum Intensity Projection which takes the maximum sample value along the ray as a pixel value.

If multiplicative compositing is employed, the ray-compositing has to take place from the start to the end of the ray-cache even though only a portion of it has new values. (I ignore the adaptive ray termination here for simplicity, though it is implemented.) Despite this, decoupling relatively expensive interpolated ray-sampling from ray-compositing saves time. The span of the ray to be composited, from the entry to the exit of the reconstruction buffer bounding box, is computed by clipping the ray to the reconstruction buffer bounding box.

There is a minor point to be said about the process above. If the shading and sampling is actually performed on the very last slab reconstructed, there can be an 'open end' effect, where the abrupt cut-off of the partially reconstructed volume shows up in the shading result and in the final image. To avoid this, the reconstruction stage and the later stages are offset by one slab; if a slab n is reconstructed, slab n-1 is shaded, sampled, and composited.

Note that the arguments above about incremental volume rendering assumed the fixed viewpoint as well as fixed shading parameters. If the viewpoint is changed, the ray-sampling and compositing have to be done

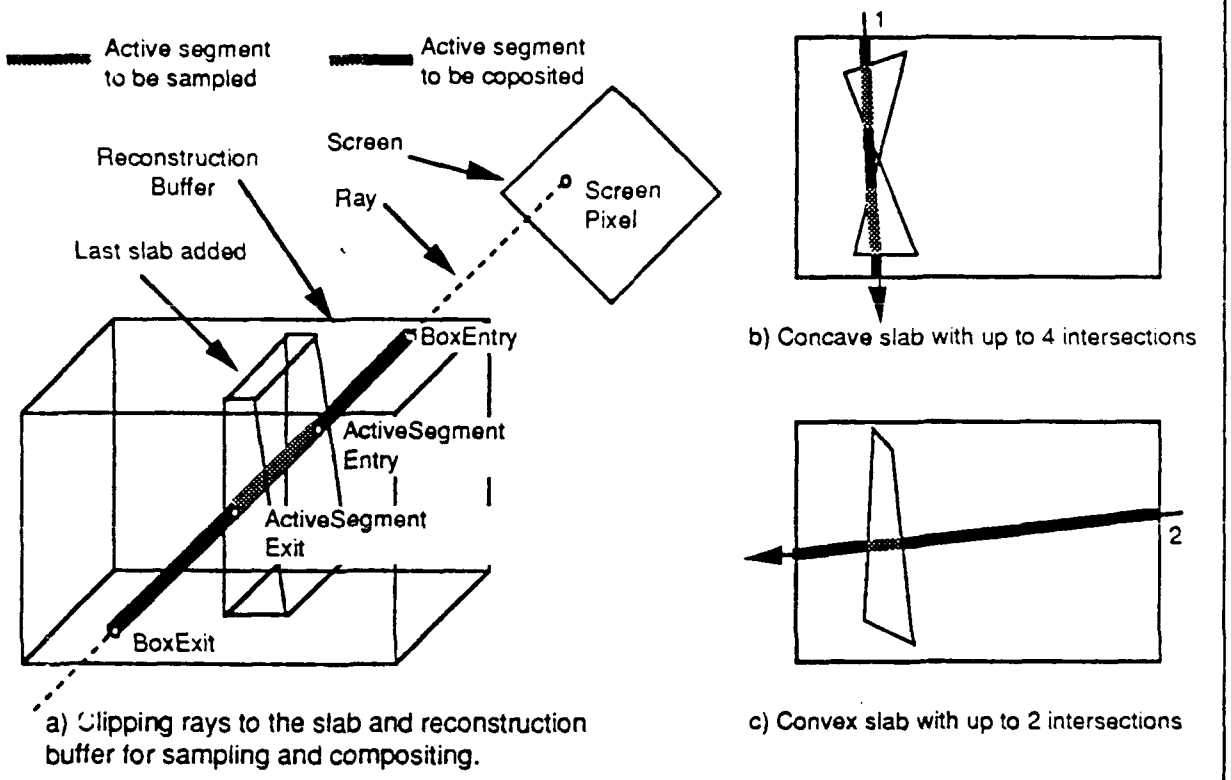


Figure 7a. (Left) A ray from a pixel is clipped to the reconstruction buffer bounding box, and to the last slab for compositing and sampling, respectively.

Figure 7b. (Right Top) Slab can be concave, with up to 4 intersections, or

Figure 7c. (Right Bottom) convex with up to 2 intersections.

essentially on the entire data. Or, if the shading parameters are changed, the shading has to be done in addition to ray-sampling and compositing. In such cases, various forms of coherences, such as image coherence and object coherence as well as temporal coherence can be used improve performance. Current single processor implementation does not include these optimizations for non-incremental cases.

Improving Rendering Performance

Our goal is to make this system as interactive as possible, running at the slowest with the image generation speed (reconstruction and rendering combined) of more than 1 frames per second on the proof-of-concept system using the Pixel-Planes 5. We have to make the rendering and reconstruction as fast as possible.

In the interactive, incremental 3DE system, the volume data set changes for every image frame. Many of the optimization schemes that assumed multiple image generation from single data set can not be applied. For example, skipping empty space by hierarchical space enumeration using octree may not be appropriate since octree is usually precomputed. Similar argument goes to

the precomputing gradient vectors or even complete shading (color and opacity) values.

Also, use of incremental scheme tipped the scale in relative costs of various stages of volume rendering. A good example is clipping the rays to the volume of interest (i.e., rectangular data set). It was a minute part of the ray-casting stage in the conventional ray-casting based volume rendering. In the incremental algorithm, since the volume to be sampled is much smaller, relative cost of ray-clipping has become a significant part of ray-casting process. Shading (Phong shading) and ray-sampling still are major parts of the rendering process as well as the reconstruction depending on the algorithm used.

Ray Clipping by Scan Conversion

In the ray-casting based incremental volume rendering algorithm, a slab of volume data is incrementally added as the input image slice arrives. The resultant volume data is then shaded, sampled, and composited. With the introduction of ray-cache, sampling the unchanged part outside of the slab is obviously wasteful. To sample rays only in the slab formed by latest two slices, each ray from the pixel is clipped to the slab. As mentioned, I used an algorithm similar to Cyrus-Beck [Cyru78] in the first implementation of incremental volume renderer. As seen in the Table 1., clipping rays to the slab has taken up large portion of rendering time. We needed a faster algorithm.
I have developed a new line-polyhedron intersection algorithm called *D-buffer* algorithm for Distance Buffer. It is not as general as the Cyrus-Beck clipper. But it is much more efficient if applied to computing intersections of non-trivial number of rays from a screen with polygons. It takes advantage of the efficient polygon scan conversion algorithm to compute intersection distances (see Figure 8a). Following is the sketch of the D-buffer algorithm for ray-clipping.

{ This routine computes the intersection of all the rays from the screen pixels with the slab defined by polygons, using modified Z-buffer algorithm. }
procedure RayClip(Slab)
begin
• Decompose slab into convex polyhedrons, if the slab is concave.
for each polyhedrons

> • Clip it to the reconstruction buffer bounding box. Clipped faces must be 'closed' by new polygons.
>
> **for each** polygon that forms the polyhedron,
>
> > • Transform it into the canonical parallel projection view volume.
> > • Clip it to the canonical view volume
> > • Scan convert the polygon into D-buffer, using PixelUpdate() for each pixel.

end;

{ Polygon scan conversion routine calls PixelUpdate per scan converted pixel (u, v, n), to update the D-buffer. (u,v) is the location of pixel on the screen (integer) while n is the screen Z value (real) of the scan converted point.}
procedure PixelUpdate(u, v, n)
begin

• Compute the Euclidian distance from projected 2D point (u,v) in 3D screen space, to its back-projected point in 3D screen coordinate (which is the intersection).
• Insert the distance for pixel (u, v) to the corresponding entry of the distance list, which is sorted by the distance.
end

In the PixelUpdate() routine, if the projection is orthogonal, scan converted screen Z value *is* the Euclidian distance from the pixel to the 3D point projected to that pixel. If the projection is perspective, the distance must be computed by back-projecting the screen pixel and taking the square root.

Computed distances from the pixel to the intersections with the polygons are kept in the distance list for each pixel (See Figure 8b). This list is kept in ascending order as an intersection is inserted. A ray has either two or four intersections with a slab, so our implementation has 4 position array for the distances. After two or four intersection distances are obtained, we have to determine the interval(s) where the ray is to be sampled. They are the distances in 3D screen space from the screen pixel to the entry and exit intersections of the slab. Here all the polyhedrons are convex, paring the intersections to intervals by determining whether an intersection is entry or exit can be made simply by the parity rule. This can be assured (in non-singular cases) by making the polyhedron closed.

To be more robust and general, 'side' of the polygon can be determined by the direction of polygon's surface normal. This is not necessary to clip rays to the slab. But knowing the side of the polygon has its use later when rendering polygonal objects using the D-buffer mechanism. Note that single check of polygon's side per polygon is enough and it is not a costly.

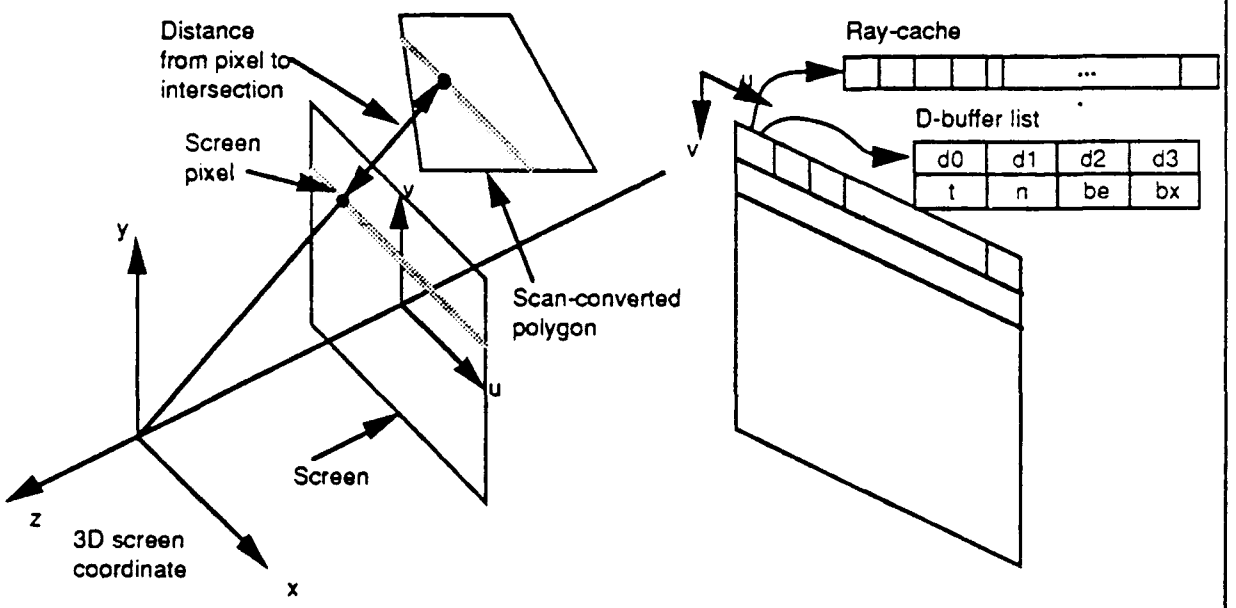


Figure 8.a Ray-polygon intersection can be computed efficiently by scan conversion.
Figure 8.b D-buffer has 1D array for ray-cache, as well as distance list for four intersections possible for the slab.

D-buffer structure has a few other fields per pixel, as seen in Figure 8b. Obviously, there is the ray-cache, a 1D array that stores sampled ray values. Interval of the ray distances to be composited are stored in be and bx pair, which are the entry and exit to the reconstruction buffer bounding box. If the ray from the pixel needs compositing is marked in field t. Numbers of intersections are stored in n as polygons are scan converted to determine number of sampling intervals.

The D-buffer algorithm above takes advantage of the geometric coherence in the polygons that consists the polyhedron being intersected, through the use of standard polygon scan conversion technique used in computer graphics [Fole90]. Use of the polygon scan conversion means that only those rays that actually intersect polygons require computation. This is in contrast to the algorithm based on conventional ray-polygon intersection algorithm where all the rays must be checked for intersection before they are rejected. Also, due to its incremental nature, for a non-trivial number of intersections, cost per ray is very small.

We have implemented the ray-clipping by D-buffer, as well as the Cyrus-Beck algorithm. Table 1. compares the execution time of various visualization stages per single image generation for three different algorithms; without ray-clipping, ray-clipping by Cyrus-Beck algorithm, and ray-clipping by D-buffer algorithm. In all the cases, reconstruction (by forward-mapping algorithm) and shading (Phong shading) are incremental. The forward mapping reconstruction algorithm used is fast but not of high quality. It is very likely that the computational complexity of the reconstruction will increase in the future as reconstruction quality is improved. The program written in C language is compiled with -O option and executed on the DEC 5810 with 256MByte of memory which is running Ultrix operating system. As an input, 36 2D image slices of roughly 5 mm interval are read from a disk file along with their geometries. 34 images of 256 x 256 pixels each (whose 34th image is similar to Picture 2) are generated but not written to the disk to discount the time to write images. Timing are measured by UNIX system call `times()` for the program to generate 34 images, which is then divided by 34 to make them average timing per image.

As seen in the table, limiting the sampling to inside the slab greatly reduced the time for ray-sampling. Furthermore, ray-clipping by D-buffer virtually removed the overhead of clipping rays to the slab. What is left in the sampling stage is the tri-linear interpolated sampling. Reducing time for the ray-sampling further will need such optimizations as skipping empty spaces, or adaptively reducing the number of rays cast. These optimization will be

Processing Stages	No Clipping	Cyrus-Beck	D-buffer
Reconstruction	0.60s (0.3%)	0.60s (3.3%)	0.59s (5.9%)
Shading	2.87s (1.4%)	2.83s (15.4%)	2.81s (28.3%)
Ray-clipping	0.00s (0.0%)	8.40s (45.8%)	0.10s (1.0%)
Ray-sampling	185.56s (91.8%)	4.75s (25.9%)	4.67s (47.1%)
Ray-compositing	13.13s (6.5%)	1.77s (9.6%)	1.75s (17.6%)
Total time	202.18s	18.35s	9.92s

Table 1. Execution time of ray sampling and compositing for the no-clipping, clipping with Cyrus-Beck clipper, and clipping with D-buffer algorithm. Both reconstruction and shading stages are done incrementally on all three cases. Numbers in parenthesis are the relative time spent in percent.

difficult to incorporate into the incremental volume rendering algorithm.

As mentioned above, the relative cost of the reconstruction may increase as the reconstruction algorithm become sophisticated. The shading stage will be more dominant with the real image taken from a human target than the doll phantom image scanned in the water tank. This is so because the empty volume where the shading computation can be abbreviated will be much less in the image from human target. Time for the compositing stage is small because only the rays with new samples, i.e., those that touch the slab, need compositing, leaving majority of the pixel untouched.

Hierarchical Ray-Cache

Ray-cache introduced before was a simple 1D array for each pixel, which caches every sample taken along the ray at unit intervals. This has reduced the time to re-generate image. Still, compositing stage has been wasting computation, by repeatedly compositing entire span inside the reconstruction buffer bounding box. This is especially so if the slab occupies only a small span out of the entire span of the ray inside the reconstruction buffer bounding box, as illustrated in Figure 7c. In this section, I introduce a new ray-cache structure to improve ray-compositing speed, which is currently being implemented along with the polygonal object rendering capability described in the following sections.

Average ray-compositing time can be improved quite a bit by saving and reusing the partially composed values in a tree structured *hierarchical ray-cache* (or HRC), instead of the linear ray cache (LRC) introduced before. It is depicted in Figure 9 for 32 entry, 4-ary tree case, though it can have any arity more than 1. This is another case of space-time trade off; HRC requires even more memory than the LRC, but gains substantially in speed.

In HRC, the tree is maintained so that each node has the values of opacity

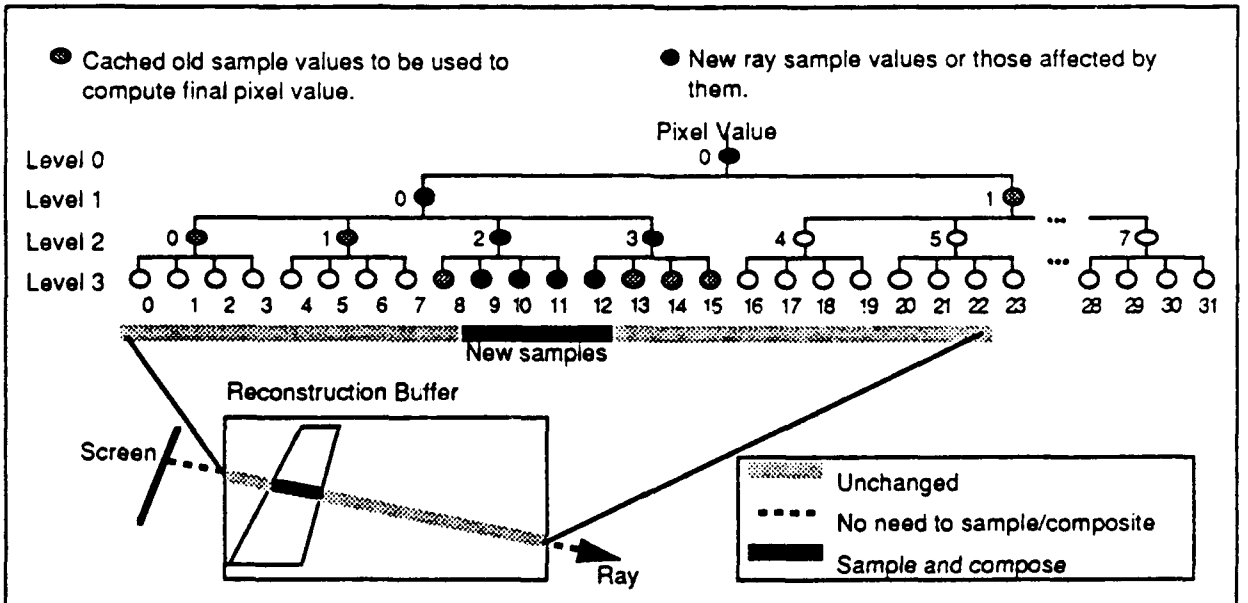


Figure 9. Hierarchical Ray Cache with 32 entries. Changes due to the insertion of a small span of samples are propagated bottom up to the root, which is the pixel value. This takes much smaller number of compositing than with the linear ray-cache.

and color that are the result of compositing those values in their child nodes from left to right order of traversal. The leaves of the tree have the opacity and color sample values as they are sampled along the ray. The root of the tree, if the property above is maintained, has the values of opacity and color as the result of correct compositing along entire ray samples, that is, the screen pixel. As a set of new sample values is added to the leaves, updating process to maintain the property takes place. The updating process of node values takes place from bottom up. For each node, if value of any of its child nodes has changed, a new pair of opacity and color values are computed by compositing. The HRC for a pixel can be embedded in a linear array, and the tree traversal can be done efficiently by index manipulations.

Assuming a binary tree, for a total ray span of 2^n samples, change in one sample requires only n number of compositing to obtain the pixel value. This is the most favorable case for the HRC and is the big improvement over 2^n-1 compositing necessary for the LRC. The worst case possible for the HRC is when the entire span of the ray inside the reconstruction buffer must be sampled anew. In that case, HRC needs the same 2^n-1 compositing as LRC, while HRC needs more memory write, 2^{n+1}-1, compared to 2^n-1 for the LRC. For most of the cases where the slabs are oblique against the ray, and are thin compared to the reconstruction buffer bounding box dimensions, HRC will be much more efficient than the LRC.

Integrating Polygonal Objects

It is obviously advantageous to have geometric object rendered together with the volume data in the same image. [Levo90b] presents two methods to achieve this in the framework of front-to-back image-order volume renderer. One is to 3D scan convert a polygon into the volume data, with appropriate filtering to band-limit the image, and volume render as a volume image. The other method, which he called a *hybrid ray-tracer*, is to combine ray-casting process for the volume data with the ray-casting for the polygons.

In Levoy's hybrid ray-tracer, combining the scan converted polygons with the volume image is done as the volume image is ray-traced in the world space. In addition to the burden of interpolated sampling of rays, adding polygonal objects interferes with such optimization as space-skipping by hierarchical space enumeration using the octree data structure. For these reasons, hybrid ray-tracer is not fast in his implementation.

In the following, I propose an algorithm that realizes very fast rendering of polygonal objects, both opaque and transparent, cut-away of volume image defined by polyhedrons, change of color or opacity of polyhedral volume for highlighting, etc., in the volume rendered image with fixed viewpoint. The efficiency (and limitation of fixed viewpoint) of this algorithm comes from the fact that it works in 3D screen space, combining viewing transformed and resampled volume data in the ray-cache with the polygonal and polyhedral objects scan converted into the D-buffer.

My algorithm is in line with the Atherton's *object buffer* algorithm [Athe81]. He described an algorithm that saves a list of all the depth (z) values scan converted, ordered by z and accompanied by each surface's attributes, in the object buffer. As far as the viewpoint is fixed, object buffer allows, in addition to standard hidden surface removal, such effects as cutaway viewing of the

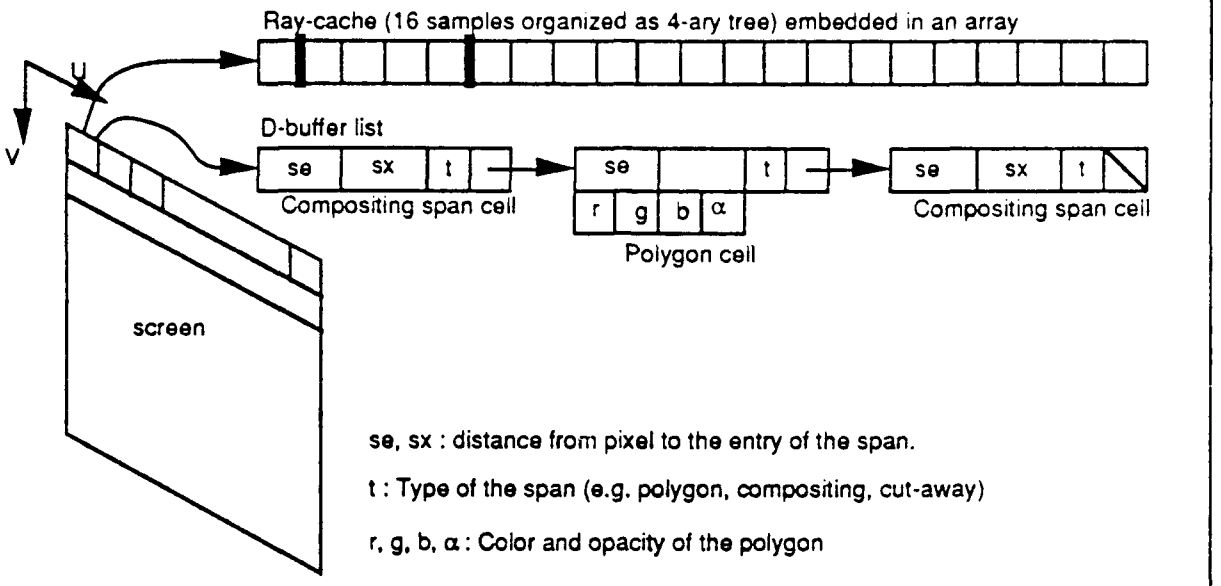


Figure 9. Structure of the D-buffer with hierarchical ray cache. 16-entry 4-ary tree ray-cache is embeded in a 1D array of size 21.

objects, selecting surface color, and transparency or translucency of the objects. Atherton also suggests the object buffer can also be *object buffer matrix*, which is essentially a voxel array, as in Ray-Cache in our incremental volume rendering algorithm. The paper suggests the application to the 3D visualization of CT and seismic analysis. More recently, [Eber90] added volume rendering capability on top of an A-buffer [Carp84] based polygon renderer. It is to combine gaseous objects and solid textures to the mostly polygonal objects. [Garr90] and [Miya90] combined volume renderer for an irregular geometry volume data from such data source as Finite Element Mesh with the z-buffer algorithm based polygon renderer.

Figure 9 shows the new frame buffer structure which combines HRC and D-buffer. D-buffer has a linked list of *span-cells*, where each span-cell has its type associated. The type field tells if the interval is a sampled volume data to be composited, a volume of interest to be rendered with modified color and/or opacity, or a polygon with associated colors and opacity. HRC's tree structure is embedded in a 1D array as a heap for compact representation and ease of traversal. Types of the span-cells are, for example,

1) Sample Span cell : Distances of the volume sample span to be composited,
2) Polygon cell : A polygon distance, color and opacity values.
3) Highlight span cell : Distance of the highlight span, with its color and opacity modification values.

There is no cell type for cut-away span, since cut-away is done by not compositing the cut-away span, through modification of the sample span cells.

Integrating Polygons

The incremental volume rendering algorithm keeps ray-samples in the ray-cache to reuse them. Thus, for a fixed viewpoint, adding and correctly compositing a polygon to the volume image involves only; 1) scan converting the polygon into the D-buffer, and 2) compositing the cached ray-samples with the scan converted polygons. If the distance value of the scan converted polygon falls inside one of the ray span, the span is split into two sections. The two sections become two separate sample cells in D-buffer, and a new polygon cell is inserted in between.

In this algorithm, there is no ray-sampling involved for image generation (of fixed viewpoint) which makes this process efficient. Furthermore, with the HRC, compositing a scan converted polygon amounts roughly to inserting a sample slice of thickness 1 sample and then updating the hierarchical ray cache.

The cost of polygon scan conversion depends on usual factors associated with polygon scan conversion, e.g., the shading models used, and the cost of inserting the polygon's cell into the D-buffer. The cost of D-buffer update will be significant if the number of cells in the D-list is large, especially with linear list structure. The cost of ray-compositing depends both on the number of rays, as well as the cost of compositing a ray. The number of rays is proportional to the area of polygons projected onto the screen. The cost of compositing a ray using hierarchical ray-cache per polygon is proportional to the logarithm of the number of ray-samples, as discussed in the preceding section.

In implementing the algorithm, there are two sources of aliasing to be taken care of. One is the aliasing at the edge of the polygon. The second is familiar in the Z-buffer algorithm, and amenable to such solutions as super sampling, or A-buffer [Carp84], and not discussed here. The other is characteristic to the volume rendering. To composite the polygon with the volume samples correctly, the amount of volumes inside a unit sample grid that is in front and back of the polygon should determine the amount of opacity contribution the polygon makes. Determining these volumes correctly is somewhat costly, and I have adopted a simpler but visually satisfactory solution as proposed in [Levo90b]. It treats the polygon locally inside the ray-sample cube as a plane perpendicular to the ray and placed at the ray-polygon intersection point. Then the opacity of the unit sample cube is divided into front and back of the polygon, α_f and α_b respectively, in proportion to the thickness of partial cube in front and back of the polygon. α_f and α_b is computed as;

$$\alpha_f(u) = 1-(1-\alpha_v(u))^{t_f^{(u)}/t_v^{(u)}}$$

and

$$\alpha_b(u) = 1-(1-\alpha_v(u))^{t_b^{(u)}/t_v^{(u)}}$$

where t_f and t_b are the thickness of the cube in front and back of the polygon, while t_v is the thickness of the unit sampling cube. $\alpha_v(u)$ is the opacity of the entire cube, while c_v is the color of the entire cube. Compositing proceeds from front to back, by compositing the portion in front using c_v and α_f first, then the

polygon as an object of 0 thickness but finite opacity α_p and color c_p, and finally the portion in the back of the polygon using α_b.

Integrating Polyhedral Volume

Cutting away a section of the volume image is a very useful tool for interaction with the image, to see objects obscured by an opaque surface, for example. This may also be used for simulated surgery. Highlighting a volume, through changing the color or opacity of a volume can be a useful tool to highlight volume of interest. For example, in radiation treatment planning, the traces of treatment beams can be highlighted while the others are kept dim but visible enough to give anatomical reference.

With D-buffer and hierarchical ray-cache, removing cutaway volumes or highlighting of volume by modified color and/or opacity is almost as simple as scan converting the polygons which define the volume, and re-compositing. Seen from the perspective of solid modeling, cut-away or highlighting of a polyhedral volume of interest in the volume data is performing Boolean operation among different (B-reps and cell enumeration) representations of objects.

A volume of interest can be highlighted by increasing or decreasing the opacity of the volume, and/or by changing the color of the volume. To do this, the polyhedron defining the volume of interest is scan converted into the D-buffer. First, for a designates span(s) of a ray, opacity and/or color of the defined volume is modified at the leaves of the hierarchical ray cache. This does not have to literally modify the value stored in the leave cells of the HRC. Just mapping the color by table, or multiplying the opacity modification factor as the HRC is updated is enough. The HRC is updated from bottom to top as if new samples are inserted to the span to be highlighted. Here, as noted in the rendering of polygons, the compositing near boundary of the object must take handle the partial volume of the unit sample cube, to correctly composite the modified polygonal volume. The cost of highlighting a volume will be higher than rendering polygons, since it tends to disturb more in the HRC, forcing more update effort.

Cutting away a section of volume defined by polyhedron can be done in done by the same mechanism as above, by simply making the opacity 0. But there is somewhat better special case approach. Instead of modifying the individual sample values, the D-buffer's span list is modified to remove the spans of the sections to be cut-away. Then, simply composite the ray-cache according to the resulting span list. The cost to cut-away a contiguous section on a ray is small; it is the comparable to adding two polygons by scan conversion into D-buffer minus shading calculations. Since actual compositing is avoided for the cut-away sections, total compositing time may even be less with volume cur-away.

I should re-iterate that the cut-away of volume and rendering of polygons into a fixed viewpoint volume image is very fast with accurate volume compositing using D-buffer and HRC. Even though highlighting volume of interest by changing color or opacity is not quite as fast, it still avoids costly ray sampling process entirely, which will provide quite responsive interaction with the volume image. Please also note that this 'late-binding' technique of integrating polygonal and polyhedral objects to the volume rendered image using hierarchical ray-caching in 3D screen space is not limited to the

incremental volume rendering. It can be applied to other conventional volume rendering algorithms based on ray-casting.

Limitation of this technique of quickly integrating the polygonal objects with the volume image is the fixed viewpoint. Polygons, cut-away volumes or highlighted volume of interest defined in the image can not be viewed from different viewpoints easily, since it exists only in the image space. To view it from different viewpoint, or to reconstruct the 'sculpting' done by polygonal objects later as may be necessary in radiation treatment planning, geometric information of these polygonal objects must be saved separately. In terms of performance, large number of intersections with polygons per ray will degrade the performance of hierarchical ray-cache, in addition to obvious overhead of scan-converting many polygons. Also, as mentioned, the linear list structure used in the D-buffer list will not be the most efficient for large number of polygons.

Conclusion

In this paper, we have reported the overview of the incremental, interactive 3D ultrasound echography system, where 2D image slices are acquired along with their locations and orientations to be reconstructed and volume rendered. The system generates a new image as a new 2D slice is acquired, to maximize interactivity. The visualization algorithm works in incremental manner, limiting the computations to the volume where the new input has arrived.

We have established the basics of the efficient incremental volume rendering algorithm, which takes advantage of the incremental nature of the input. We have designed and implemented a faster ray clipping algorithm using polygon scan conversion to clip rays to the slab where the ray should be sampled. The new ray clipping algorithm showed marked improvement over the method we have used before.

We have proposed a new ray-caching mechanism called hierarchical ray-cache to speed up the ray-compositing further. We have also proposed an algorithm to render polygonal objects quickly in screen space composited correctly with the volume image. With this proposed algorithm, various operations on the volume image, such as cut-away of a polyhedral volume, insertion of polygons, and highlighting polyhedral volumes will be possible at an interactive rate.

Clearly, we need more work to get to our goal of interactive acquisition 3D visualization system. First, economical reconstruction algorithms with good reconstruction quality have to be developed to reconstruct irregular input slices. Current bottlenecks, the shading stage and the ray-sampling stage have rooms for performance improvements. Some variants of space skipping by enumerating empty space to reduce number of sampling may be applicable to the incremental volume rendering. At this stage, image adaptive ray-casting such as the one in [Levo90a] seems hard to adapt to the incremental volume rendering.

We are planning to parallelize the algorithm to be run on the graphics oriented heterogeneous multicomputer Pixel-Planes 5. We have experimented a distributed volume rendering algorithm on a set of workstations, which showed promising result. It was parallelized in images space, and based demand-paged distributed shared memory model, similar to [Bado90]. For the

incremental, interactive 3DE system on Pixel-Planes 5, I am planning to use data parallelism in object space for the reconstruction and shading, and the data parallelism in image space for the ray-sampling and compositing stages. We expect to have proof of concept system with Pixel-Planes 5 running in 1991.

Acknowledgment

The authors would like to thank Vern Katz, M.D. for the use of the ultrasound scanner, and Jeff Butterworth for developing the image acquisition program. We also would like to thank the Wake Radiology Association for donating ROHNAR 5580 to us. This research is supported by NSF grant number CDR-86-22201.

Reference

[**Athe81**] Atherton, P. R. (1981). "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display." ACM Computer Graphics. **15**(3): 279-287.

[**Bado90**] Badouel, D., Bouatouch, K., Priol, T.,. (1990). Ray Tracing on Distributed Memory Parallel Computers : strategies for distributing computation and data. Institut De Recherche en Informatique et Systèmes Aléatoires, Universitaire de Beaulieu, France, Technical Report 508

[**Bill90**] Billion, A. C. (1990). Phillips Paris Research Lab. Personal Communication.

[**Brin78**] Brinkley, J. F., Moritz, W.E., and Baker, D.W. (1978). "Ultrasonic Three-Dimensional Imaging and Volume From a Series of Arbitrary Sector Scans." Ultrasound in Med. & Biol. **4**: 317-327.

[**Carp84**] Carpenter, L. (1984). "The A-buffer, an Antializsed Hidden Surface Method." ACM Computer Graphics. **18**(3): 103-108.

[**Cyru78**] Cyrus, M., and Beck, J. (1978). "Generalized Two- and Three-Dimensional Clipping." Computers and Graphics. **3**(1): 23-28.

[**Eber90**] Ebert, S. D., and Parent, R.E. (1990). "Rendering and Animation of Gaseous Phenomena by Combining Volume and Scanline A-buffer Technique." ACM Computer Graphics. **24**(4): 357-366.

[**Fole90**] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). Computer Graphics Principle and Practice. Addison-Wesley Systems Programming Series. Addison-Wesley. 2'nd Edition,

[**Fran82**] Franke, R. (1982). "Scattered Data Interpolation : Tests of Some Methods." Mathematics of Computation. **38**(157): 181-200.

[**Fuch89**] Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molner, S., and Israel, L. (1989). "Pixel Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories." Computer Graphics. **23**(3): 79-88.

[**Garr90**] Garrity, M. P. (1990). "Raytracing Irregular Volume Data." ACM Computer Graphics. **24**(5): 35-40.

[**Ghos82**] Ghosh, A., Nanda, C.N., and Maurer, G. (1982). "Three-Dimensional Reconstruction of Echo-Cardiographics Images Using The Rotation Method." Ultrasound in Med. & Biol. 8(6): 655-661.

[**Hott89**] Hottier, F. (1989). Phillips Paris Research Lab. Personal Communication.

[**Lalo89**] Lalouche, R. C., Bickmore, D., Tessler, F., Mankovich, H.K., and Kangaraloo, H. (1989). "Three-dimensional reconstruction of ultrasound images." SPIE'89. **Medical Imaging**: 59-66.

[**Levo88**] Levoy, M. (1988). "Display of Surface from Volume Data." IEEE CG&A. **8**(5): 29-37.

[**Levo89**] Levoy, M. (1989). Display of Surfaces From Volume Data. Ph.D Thesis, University of North Carolina at Chapel Hill, Computer Science Department

[**Levo90b**] Levoy, M. (1990). "A Hybrid Ray Tracer for Rendering Polygon and Volume Data." IEEE CG&A. **10**(2): 33-40.

[**Levo90a**] Levoy, M. (1990). "Volume rendering by adaptive refinement." Visual Computer. **6**: 2-7.

[**McCa88**] McCann, H. A., Sharp, J.S., Kinter, T.M., McEwan, C.N., Barillot, C., and Greenleaf, J.F. (1988). "Multidimensional Ultrasonic Imaging for Cardiology." Proc.IEEE. **76**(9): 1063-1073.

[**Mills90**] Mills, P. H., and Fuchs, H. (1990). "3D Ultrasound Display Using Optical Tracking." 1'st Conference on Visualization for Biomedical Computing. Atlanta, GA, 490-497.

[**Miya90**] Miyazawa, T. (1990). "A high-speed integrated rendering for interpreting multiple variable 3D data." SPIE, To appear. :

[**Naka84**] Nakamura, S. (1984). "Three-Dimensional Digital Display of Ultrasonograms." IEEE CG&A. **4**(5): 36-45.

[**Neem90**] Neeman, H. (1990). "A Decomposition Algorithm for Visualizing Irregular Grids." ACM Computer Graphics. **24**(5): 49-56.

[**Nikr84**] Nikravesh, P. E., Skorton, D.J., Chandran, K.B., Attarwala, Y.M., Pandian, N., and Kerber, P.E. (1984). "Nikravesh, P.E., Skorton, D.J., Chandran, K.B., Attarwala, Y.M., Pandian, N., and Kerber, P.E." Untrasonic Imaging. **6**: 48-59.

[**Novi90**] Novins, K. L., François, X. S. and Greenberg, D. P. (1990). "An Efficient Method for Volume Rendering using Perspective Projection." ACM Computer Graphics. **24**(5): 95-102.

[**Ohbu90**] Ohbuchi, R., and Fuchs, H. (1990). "Incremental 3D Ultrasound Imaging from a 2D Scanner." First Conference on Visualization in Biomedical Computing. Atlanta, GA, 360-367.

[**Raic86**] Raichelen, J. S., Trivedi, S.S., Herman, G.T., Sutton, M.G., and Reichek, N. (1986). "Dynamic Three Dimensional Reconstruction of the Left Ventricle From Two-Dimensional Echocardiograms." Journal. Amer. Coll. of Cardiology. **8**(2): 364-370.

[**Sabe88**] Sabella, P. (1988). "A Rendering Algorithm for Visualizing 3D Scalar Fields." Computer Graphics. **22**(4): 51-58.

[**Shat84**] Shattuck, D. P., Weishenker, M.D., Smith, S.W., and von Ramm, O.T. (1984). "Explososcan: A Parallel Processing Technique for High Speed Ultrasound Imaging with Linear Phased Arrays." JASA. **75**(4): 1273-1282.

[**Shir90**] Shirley, P., and Tuchman, A. (1990). "A polygonal approximation to direct scalar volume rendering." ACM Computer Graphics. **24**(5): 63-70.

[**Stick84**] Stickels, K. R., and Wann, L.S. (1984). "An Analysis of Three-Dimensional Reconstructive Echocardiography." Ultrasound in Med. & Biol. **10**(5): 575-580.

[Upso88] Upson, C., and Keeler, M. (1988). "VBUFFER: Visible Volume Rendering." ACM Computer Graphics. **22**(4): 59-64.

[Wilh90] Wilhelms, J., Challinger, J., and Vaziri, A. (1990). "Direct Volume Rendering of Curvilinear Volumes." ACM Computer Graphics. **24**(5): 41-47.